Enhancing the experience of users regarding the email classification task using labels

Marcelo G. Armentano and Analía A. Amandi

ISISTAN Research Institute, UNICEN-CONICET
Campus Universitario, Paraje Arroyo Seco, Tandil, 7000, Argentina
{marcelo.armentano, analia.amandi}@isistan.unicen.edu.ar

Abstract. Email is an indispensable tool for communication and users might have to deal with large volumes of information which they can not always operate efficiently. For these users, the organization of emails is a tedious task. The use of automatic filters is not always possible or effective, because of difficulties regarding how to create a specific rule or because their use is impractical in some situations. In this article, we present an approach to enhance a webmail client with an interface agent that helps the user to label incoming email based on the knowledge of the user's preferences. We not only considered the label that can be applied to different emails but also how to better interact with the user to provide him/her with assistance in the labeling procedure. We performed a set of experiments using Google's webmail system, Gmail, obtaining a good rate of acceptance of the agent interactions.

1 Introduction

The electronic mail is nowadays one of the most used and effective communication mean and an indispensable working tool in many companies. The type of information that users received by email varies from user to user but also each user receives information from different projects, activities, interests, social networks, games, advertisements, etc. When the emails received by a user grows in number and diversity, grouping them becomes a necessary task to facilitate the order and the reading of the information received. The task of manually classifying incoming emails takes a considerable amount of time to users. Email clients commonly offer different tools to facilitate the management of incoming messages, for example, grouping related messages into *threads*, following the idea that there exist a conversation between a message and its successive replies. Another example is the use of folders or labels to manually classify incoming emails. In this latter case, email clients often provide the possibility of creating user-defined *filters* to apply a certain label to a message or to move it to a determined folder according to certain preset rules. However, filters are not always effective or possible to apply, either because the user does not have enough knowledge about how to create and update them or because there are too many filters to create and its use is impractical.

The exposed above suggest the utility of having an email client with the ability to personalize the task of incoming emails classification. Personalization aims to achieve the user's satisfaction by recognizing his/her preferences and needs. In contrast to *customization*, in which the user itself adapts the application to his/her specific needs, *personalization* refers to automatically adapting an application to the specific needs of a user, using the knowledge obtained by analyzing the user behavior and the data generated by him/her. The personalization process is initiated and conducted by the system, that continuously monitors the user behavior to automatically adapt itself. This adaptation is done without the need of the user to control how the system adjusts its behavior.

There are several mechanisms aiming to achieve the personalization of a system, particularly we are interested in the use of interface agents (also know as personal assistants). Interface agents are computer systems designed to provide personalized assistance to users who perform tasks using other software applications. An interface agent has the ability to learn the interests, preferences, priorities, objectives and needs of a user to provide proactive and reactive support in order to increase their productivity. They also serve as intermediaries between the user and a software application; they offer advice in real time, automate repetitive tasks, and hide the complexity of the system. A commonly used metaphor to understand the paradigm of interface agents is to compare them to a human assistant who works with the user in the same environment [Maes, 1994]. They have also been used, as assistants in electronic commerce [McBreen and Jack, 2001, Lieberman and Wagner, 2003], virtual teachers [Lester et al., 1997, Amandi et al., 2003, Shaoyun and Yu, 2011, Silva Logroño et al., 2012], web search assistants [Armentano et al., 2006, Armentano and Amandi, 2013], etc.

When developing agents that assist users we should pay special attention to two key issues: how to better interact with each individual user, and how to provide the right kind of assistance at the right time [Schiaffino et al., 2010]. It is natural to think that every user interacts in a personal way with his/her interface agent. That is, the action expected from the agent, the kind of errors tolerated, and the type of assistance required, vary from one user to another. For example, a given user may not want to be interrupted with notifications or suggestions. Another user may probably disapprove certain types of assistance, which means that he/she will never tolerate a certain behavior of the agent.

To fulfill the user's expectations, the agent has to observe and analyze the user reactions concerning the various assistance types and find out what assistance type they prefer in different situations. Once the agent has learned the type of assistance that the user needs, it must learn how to provide it, that is interrupting the user or not. Most users tolerate interruptions by the agent if the situation is relevant to them. Thus, the agent has to analyze the relevance of the situation before interrupting the user, probably depending on the task that the user is performing. It is also important to analyze the user's tolerance to errors that the agent can commit, providing mechanisms to enable the user to provide a simple explicit feedback about the behavior of the agent.

The consequences of not meeting the user's expectations are usually highly negative for an agent interface. When an agent makes mistakes, especially in early stages of the learning process, it determines the user's trust regarding the use of the agent. In many cases, the user may choose to completely ignore or disable the agent [LeeTiernan et al., 2001].

In this article we present *Glabel*, an approach to enhance a webmail client with an interface agent that helps the user to label incoming email based on the knowledge of the user's preferences. Besides the prediction of the label that the user might apply to each incoming email, we put special attention on how to better interact with the user to assist him/her in the tagging procedure. *Glabel* was created to make it easier for webmail users to enhance their experience regarding the email classification task using labels.

The rest of this paper is organized as follows. Section 2 presents some related work in the area of email classification. Sections 3 describes the classic confidence-based approach traditionally used by interface agents, along with its disadvantages. Section 4 presents our approach to personalizing the emails classification task and Section 5 the experiments carried out to validate our approach. Finally, in Section 6 we present our conclusions.

2 Related Work

The definition of rules or filters is the most common tool provided to the user to automatically classify incoming emails. In this approach, the user is responsible for the definition of a set of conditions that, when fulfilled by an incoming email, triggers some action. This action can be, for example, to apply a label or to delete the message. This kind of approach, implemented by most email clients, can be considered semi-autonomous because the user has to manually detect different situations and design the corresponding rules. However, the definition of rules is not appropriate for big volumes of emails that can involve different and probably overlapping concepts. Furthermore, this approach does not adapt to changes in the user's habits for email classification or in adapting to new situations for which the user has to design new rules.

Regarding autonomous applications, several approaches aim at grouping messages in subject-based folders starting from a set of incoming messages (unsupervised approaches). Automatic Mail Category Organizer [Manco et al., 2002], for example, clusters messages sharing similar features into different folders using clustering and pattern discovery techniques for mining structured and unstructured information. Cutting et al. [Cutting et al., 1992] uses a complex intermixing of iterative partitional clustering and an agglomerative scheme. Agrawal et al. [Agrawal et al., 2000] produce only cluster digests for topic discovery, and perform a message partitioning on the basis of such digests using a Bayesian classifier.

On the other hand, several learning techniques has been used to the task of classifying emails, specially to detect spam messages. Among the supervised approaches, we can mention the use of rule-based systems [Cohen, 1996], Support-

Vector Machines [Drucker et al., 1999, Kiritchenko and Matwin, 2011, Yoo et al., 2011], Bayesian networks [Sahami et al., 1998, Androutsopoulos et al., 2000, Sakkis et al., 2001, Isozaki et al., 2005], memory-based reasoning [Segal and Kephart, 1999, Delany et al., 2005], decision trees [Youn and McLeod, 2007], linear logistic regression [Aberdeen et al., 2010], neural networks [Yu and Zhu, 2009] and semantic analysis methods [Park and An, 2010].

Other approaches to categorize email messages include Collaborative filtering techniques [Jennings and Higuchi, 1993, Resnick et al., 1994], Social network analysis [Yelupula and Ramaswamy, 2008] and Search operator suggestions [Dredze et al., 2009].

There are other aspects of personalization that has been considered in other domains different to email classification, for example the device with which users access a given system. It is very common that users access an online system from different devices, such as mobile phones, notebooks, or tablets. Devices used by mobile users are diverse and heterogeneous, with different screen sizes, memory, connection speed, and computational power. Kao-Li et al. [Kao-Li et al., 2011] developed a new social tag-based method for the recommendation of multimedia items which considered the user location, audience, mobile device, and network condition. These context descriptors were used to develop a set of rules to re-rank the recommendation list derived from the user preferences.

Differently to previous approaches that concentrate their efforts on maximizing the accuracy on the label (or folder) prediction, we focused on considering how to better interact with the user when the assistant detects an opportunity to apply a label to an incoming email. Therefore, to consider whether an interaction of the agent is correct or not, not only the content of the label suggested has to be correct but also the action taken by the agent has to be the action expected by the user. In this direction, some algorithms have been proposed to decide which action an agent should execute next. Most of these algorithms are based on confidence values attached to different actions [Maes, 1994, Kozierok and Maes, 1993]. However, these works do not consider a user's interaction preferences, the possibility of providing different types of assistance, or the particularities of the situation at hand.

In the following sections, we describe the classic confidence-based approach traditionally used by interface agents, along with its disadvantages. Next, in Section 4 we present our approach.

3 Confidence-based approach

The confidence-based approach traditionally used by interface agents, is based on the confidence on an agent action. The confidence on an action indicates how sure the agent is about executing that action, and it is computed according to the agent's experience in assisting the user. For example, in [Maes, 1994] the agent computes the confidence on the prediction of an action to the current situation taking into account how many similar situations the agent has memorized,

Algorithm 1 High level decision making algorithm

Input: A problem situation Sit to deal with
Ouput: The agent has executed an action to deal with Sit
1: Select action A via learning techniques to deal with Sit
2: Compute confidence value C for A
3: **if** $C \geq do-it$ threshold **then**
4: Perform action A
5: **else if** $C \geq tell-me$ threshold **then**
6: Suggest action A
7: **else**
8: Do nothing
9: **end if**

whether or not all the nearest neighbors of the situation recommend the same action, and how close or distant these nearest neighbors are.

In the confidence-based approach, interface agents have generally three possibilities when they want to assist a user: executing a task autonomously, suggesting the user what to do, and doing nothing. These agents use two threshold values to take decisions, which are established by the user to control the agent's behavior: *do-it* threshold and *tell-me* threshold. If the confidence value associated with an agent action is smaller than the *tell-me* threshold the agent does nothing; if the confidence value is greater than the *tell-me* threshold but smaller than the *do-it* threshold, the agent tells the user what it thought he/she would do and it waits for confirmation to automate the action; and if the confidence value is greater than the *do-it* threshold the agent executes the task autonomously on the user's behalf, sending him/her a report. The *do-it* threshold is higher than the *tell-me* threshold. If the agent does not execute an action, then the user has to deal with the situation at hand, and the agent observes his/her behavior to learn from it.

Algorithm 1 shows a high level decision making algorithm, which is an abstraction of the algorithms used by the interface agents built under the confidence-based approach.

The confidence-based approach has several problems. The main one is that confidence values do not consider the way in which the user wants to interact and work with his/her agent. These agents do not take into account when the user wants each type of assistance action. They do not take into account when the user wants a suggestion, when he/she only wants a warning about a problem, when he/she wants the agent to execute an action on his behalf, or when the user does not want any assistance at all. The decision making algorithm should select the assistance action the user expects and will accept. Thus, despite the agent is able to find a good solution to a given problem it also has to analyze whether the user wants to be informed about it or not.

The interface agent has to consider not only the confidence on the assistance to be provided, but also on the type of the assistance. For example, a email labeling agent might know what label to apply for an incoming mail, but probably

the user does not want it to make the suggestion or to apply the label on his behalf. On the other hand, despite the agent probably is not confident enough to make a suggestion, the user might prefer it instead of no assistance at all.

In summary, the problem with the current action selection or decision making algorithms is that they do not take into account how the user wants to be assisted and how he/she prefers to interact with the agent in different contexts. User assistance and user-agent interaction should be personalized and contextualized in order to assist users as they expect. In consequence, the relationship between the user and the agent will be enhanced.

In this work, we propose a new solution to decide how a labeling agent should better interact with the user. Our approach takes into account not only the agent confidence on the various assistance actions, but also the user's requirements and preferences regarding these assistance actions. Thus, when the agent has to decide among various actions it will consider how the user wants to be assisted in the particular situation the agent is dealing with. We describe our proposed approach next in Section 4.

4 Glabel

Glabel is an intelligent agent based approach designed with the objective of enhancing webmail users experience regarding the email classification task using labels. Figure 1 shows the life-cycle of an interaction of *Glabel*.

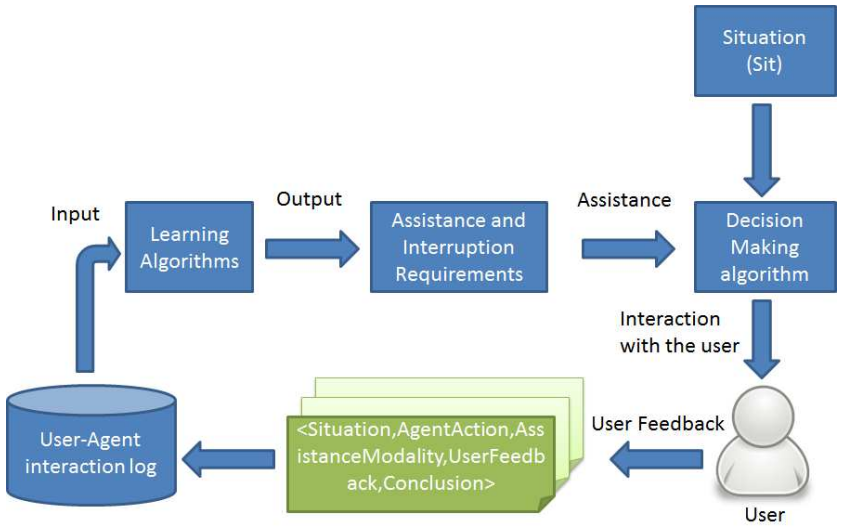


Fig. 1: Glabel learning scheme

To obtain the components of the user profile, *Glabel* first needs to record the user interactions with the agent by observing the user behavior and by considering the user feedback provided after each assistance event. We describe the components of the User-Agent interactions log in Section 4.2. Then, *Glabel* uses the information in the user profile to obtain the assistance and interruption requirements. To this aim, the User-Agent interactions log is analyzed using two profiling methods known as WATSON and IONWI [Schiaffino et al., 2010]. WATSON learns a user's assistance preferences, that is, when a user prefers a suggestion, a warning, an automated action or no assistance. IONWI learns a user's interruption preferences, that is, when a user prefers an interruption or a notification. These algorithms are described in Section 4.3. Finally, the assistance and interruption requirements are used to decide the action to perform and the interruption modality to use when facing a particular situation. The decision making algorithm is detailed in Section 4.4. Once the agent interacts with the user with the selected assistance, it obtains explicit or implicit feedback from the user. This new interaction is stored as a new experience that will be used in the future to update the knowledge the agent has about the user.

4.1 User profiles

In order to personalize the assistance provided to the user, an interface agent needs some knowledge about him/her. This knowledge is contained in a user profile. A profile is a description of the user that contains the important or interesting facts about him/her. One aspect that is of particular interest in this work is the personalization of the interaction between the agent and the user. Following the approach proposed in [Schiaffino et al., 2010], the user profile should included: information about the habits and styles of interaction with the agent, information about assistance needs in different contexts, and information on the user reactions to different types of actions of the agent (warnings, suggestions, interruptions, etc.). Formally,

$$UserProfile ::= PersonalInformation + AssistanceReq + InterruptionReq$$

The assistance requirements are represented in the user profile as a set of tuples containing a situation, the action required for that situation, and a parameter indicating the confidence with which the agent should perform that action in that particular situation:

$$AssistanceReq ::= Situation + Action + Confidence$$

The situation describes a particular context in which the assistance requirement is applied. It can be either a problematic situation in which the user might need assistance or a situation in which the agent can give and advice or suggestion. Each situation is described with a set of attributes, each of which can take a value from a predefined set. Formally, a situation is defined as:

$$Situation ::= (attribute_i, value_{ij})^+$$

For a given situation, the confidence value refers to the degree of certainty with which the agent will perform a given action. This value, ranging between 0 and 1, is computed according to the past experiences of the agent assisting the user.

$$Confidence ::= [0..1]$$

Roughly speaking , an agent has three alternatives for assisting a user: to perform an action autonomously, to suggest the user what to do next or to do nothing. To decide what alternative to take, two thresholds are usually used: *do-it* threshold and *tell-me* threshold [Maes, 1994]. If the confidence value associated to an agent action is under the *tell-me* threshold, the agent performs no action. If the confidence value is over the *tell-me* threshold but under the *do-it* threshold, the agent suggest the action to the user or simply gives a warning. Finally, if the confidence value is over the *do-it* threshold, the agent autonomously perform the action on behalf of the user. Then,

$$AssistanceAction ::= warning \,|\, suggestion \,|\, action \,|\, noaction$$

The last component of our user profiles are the interruption requirements, defined as a set of situations each of which has an associated assistance modality that can be either to interrupt or not to interrupt the user in the moment of the assistance:

$$InterruptionReq ::= Situation + AssistanceModality + Confidence$$

$$AssistanceModality ::= interrupt \,|\, notify$$

4.2 User interactions log

To build a user's profile, the interface agent must register each user interaction. According to the exposed in Section 4.1, there are several aspects of an interaction that we need to register: the situation or context in which the interaction took place, the assistance action that the agent performed, the user feedback (either implicit or explicit), how the assistance was provided (interrupting the user or not) and a conclusion on the interaction with the user (whether it was successful or not). We describe each of these items as follows:

$$Interaction ::= Situation + AgentAction + AssistanceModality + UserFeedback + Conclusion$$

- Situation: describes the situation of interest underlying the interaction between the user and the agent. The attributes describing a situation correspond to the different fields conforming an email, for example:

$$\begin{aligned} Situation ::= & (sender, user1@domain.com), (to, user2@domain.com), \\ & (cc, user3@domain.com), (bcc, user4@domain.com), \\ & (subject, "Lorem\ Ipsum"), (body, "Lorem\ Ipsum.\ Lorem\ Ipsum"), \\ & (isreply, true), (isforward, false), \\ & (hasattachments, false), (label, label1) \end{aligned}$$

- *AgentAction*: represents the action that the agent performed to face with a problem that it has to solve. The agent's action has an *AssistanceAction* and a content.

$$AgentAction ::= AssistanceAction + Content$$

  The *AssistanceAction*, as described in Section 4.1, can be a suggestion, a warning, the execution of an action or to do nothing. The content of the agent's action consists in the label that the agent selects to suggest or to apply to a message based on the user's profile.
- *AssistanceModality*: indicates how the agent will provide assistance to the user, either interrupting his/her current task or without interrupting.
- *UserFeedback*: the user can explicitly evaluate the performance of the agent from its graphical interface. However, since many users are not willing to provide feedback or consider it intrusive, this kind of feedback is optional. Nevertheless, the agent observes the user's successive actions to obtain implicit feedback. The user feedback is threefold: feedback related to the type of assistance provided by the agent, feedback related to the assistance modality, and feedback related to the content of the assistance itself.

  $UserFeedback ::= AssistanceTypeFeedback + AssistanceModalityFeedback + ContentFeedback$

- *Conclusion*: once the agent collected information about the interaction, it has to evaluate whether this interaction was successful, failed or undefined

$$evaluation ::= success \mid failure \mid undefined$$

4.3 Learning algorithms

In order to obtain the assistance and interruption requirements from a set of interaction experiences, *Glabel* uses different learning algorithms to build three classifiers: (1) to determine the content of the interaction assistance (i.e. the label to apply to a given message), (2) to determine the assistance action, and (3) to determine the interruption modality.

Each of these classifiers is based on the Naïve Bayes classifier. Naïve Bayes classifiers is a simplified version of a Bayesian Network that assumes that all the attributes are independent one from another. Despite this assumption, it has been demonstrated that Naïve Bayes classifiers are highly effective[Langley et al., 1992, Domingos and Pazzani, 1997]. Differently to other classifiers, Naïve Bayes classifier are easy to construct, requiring a time linear to the number of attributes and the number learning examples. This time complexity is essentially optimal: any learning algorithm that examines every attribute value of every training example must have the same or worse complexity [Elkan, 1997].

Algorithm 2 describes how the first classifier is built. The algorithm selects the subset of attributes from the interaction experience that will help the agent to determine the label of each email message. These attributes mainly include the *from* and *to* fields, and the corpus (that integrates the subject and body) of

Algorithm 2 Algorithm that builds different classifiers to infer the content of an action given a situation

Input: A set of interaction experiences between the user and the agent $Ex =< Sit, Act, Mod, UF, E >$ where Sit is the situation, Act is the assistance action, Mod is the interruption modality, UF is the user feedback and E is the evaluation

Ouput: A set $CNB_{languages}$ of Naïve Bayes classifiers representing the knowledge acquired with respect to the content of the assistance actions

1: $Ex_{new} \leftarrow$the subset of experiences from Ex filtering out the attributes not related to the selection of the assistance action Act
$Ex_{new} =< Sit, E >=< from, to, corpus, label, language, date, E >$
2: $languages \leftarrow$set languages in the instances Ex
3: **for all** $lang$ in $languages$ **do**
4: $Ex_{new,language} \leftarrow$set of instances from Ex with the attribute $language = lang$
5: $process(Ex_{new,language}, corpus)$ */*remove stop-words and apply a stemming algorithm to the attribute corpus*/*
6: $CBN_{language} \leftarrow$a Naïve Bayes classiffier built using $Ex_{new,language}$ as training instances, using the attribute *label* as the class attribute
7: **end for**

Algorithm 3 Algorithm that builds different classifiers to infer the content of an action given a situation

Input: A set of interaction experiences between the user and the agent $Ex =< Sit, Act, Mod, UF, E >$ where Sit is the situation, Act is the assistance action, Mod is the interruption modality, UF is the user feedback and E is the evaluation

Ouput: A Naïve Bayes classifier CNB_{type} representing the knowledge acquired with respect to the type assistance preferences of the user

1: $Ex_{new} \leftarrow$the subset of experiences from Ex filtering out the attributes not related to the selection of the assistance action Act
$Ex_{new} =< Sit, Act, E >=< from, to, label, type, E >$
2: $CBN_{type} \leftarrow$a Naïve Bayes classifier built using Ex_{new} as training instances, using the attribute *type* as the class attribute

the email. The agent also needs to determine the language of the email in order to apply the corresponding stop-words and stemming filters, assuming that each email is written in only one language. The result of the algorithm is a set of classifiers, one for each language, that predicts the label the user might apply to a given new email.

Algorithm 3 describe the procedure followed to build the classifier that given a new situation predicts the action that the agent will take. This algorithm only considers a subset of attributes of the interaction experiences. The attribute *corpus*, for instance, that was of vital importance to predict the content of the assistance, was not considered to predict the type of assistance it did not lead to good results in our experiments.

The last classifier predicts the interruption modality for a given situation. Algorithm 4 illustrates this procedure.

Algorithm 4 Algorithm that builds different classifiers to infer the content of an action given a situation

Input: A set of interaction experiences between the user and the agent $Ex =< Sit, Act, Mod, UF, E >$ where Sit is the situation, Act is the assistance action, Mod is the interruption modality, UF is the user feedback and E is the evaluation

Ouput: A Naïve Bayes classifier $CNB_{modality}$ representing the knowledge acquired with respect to the interruption modality preferences of the user

1: $Ex_{new} \leftarrow$the subset of experiences from Ex filtering out the attributes not related to the selection of the interruption modality Mod
$Ex_{new} =< Sit, Act, Mod, E >=< from, to, label, type, modality, E >$

2: $CBN_{modality} \leftarrow$a Naïve Bayes classifier built using Ex_{new} as training instances, using the attribute *modality* as the class attribute

Regarding the user profile, the classifier that predicts the content of the agent interaction, that is the label to apply to a given message, is part of the standard user profile, while the classifiers that predict the assistance type and interruption modality correspond to the enhanced user profile with assistance and interruption requirements.

4.4 Decision Making algorithm

Whenever the user receives a new email, the assistance and interruption requirements, along with the content of the assistance is inferred using the Naïve Bayes classifiers for the language in which the email is written. A confidence level is also returned by the classifiers. This confidence level is used as threshold to act on behalf of the user, suggest the action to be performed to the user or simply give the user a warning. Algorithm 5 illustrates this procedure.

First, the decision making algorithm determines the language of the incoming email (line 1). Second, it infers the most appropriate label, according to the incoming email, using the corresponding classifier $CNB_{languages}$ (lines 2 and 3). The confidence of the action is also returned by the classifier (line 4). Then, we use the CNB_{type} classifier to infer the most appropriate type of assistant, given the information of the incoming email (line 4). According to the type of assistance and the pre-configured threshold for giving suggestions, warnings or doing nothing (refer to Section 4.1), the algorithm determines what type of assistance the agent will give to the user (lines 6-14). Finally, if the assistant type is different that "no-action", we have to determine how to interact to the user: interrupting or notifying (lines 15-22)

5 Experimental evaluation

5.1 Integration with an email client

For experimental proposes, *Glabel* was implemented to be integrated with Google's web email client, Gmail©, using a Client-Server architecture (Figure 2).

Algorithm 5 Decision Making algorithm

Input: A situation Sit representing the arrival of a new email and the user profile consisting in the standard user profile and the interaction user profile $< CNB_{languages}, CNB_{type}, CNB_{modality} >$

Ouput: The agent has taken a decision Dec involving the label to apply, the type of assistance and the modality of the interaction.

1: $language \leftarrow$infer the language of the new email Sit
2: $Dec.action \leftarrow$infer the suggested action for Sit using $CNB_{languages}[language]$
3: $Sit.label \leftarrow Dec.action$
4: $conf \leftarrow$the confidence associated with the content of the action
5: $type \leftarrow$infer the action type for Sit using CNB_{type}
6: **if** $type =$ "$action$" AND $conf \geq$"$action\ threshold$" **then**
7: $Dec.type \leftarrow action$
8: **else if** $conf \leq$"$action\ threshold$" OR $type =$ "$suggestion$" AND $conf \geq$"$suggestion\ threshold$" **then**
9: $Dec.type \leftarrow suggestion$
10: **else if** $type =$ "$warning$" AND $conf \geq$"$warning\ threshold$" **then**
11: $Dec.type \leftarrow$"$warning$"
12: **else**
13: $Dec.type \leftarrow$ "$no - action$"
14: **end if**
15: **if** $Dec.type \neq$ "$no - action$" **then**
16: $modality \leftarrow$infer the modality of the assistance for $< Sit, Dec.type >$ using $CNB_{modality}$
17: **end if**
18: **if** $Dec.type \neq$ "$interrupt$" AND $modality.confidence <$ "interruption threshold" **then**
19: $Dec.modality \leftarrow$ "$notification$"
20: **else**
21: $Dec.modality \leftarrow modality$
22: **end if**

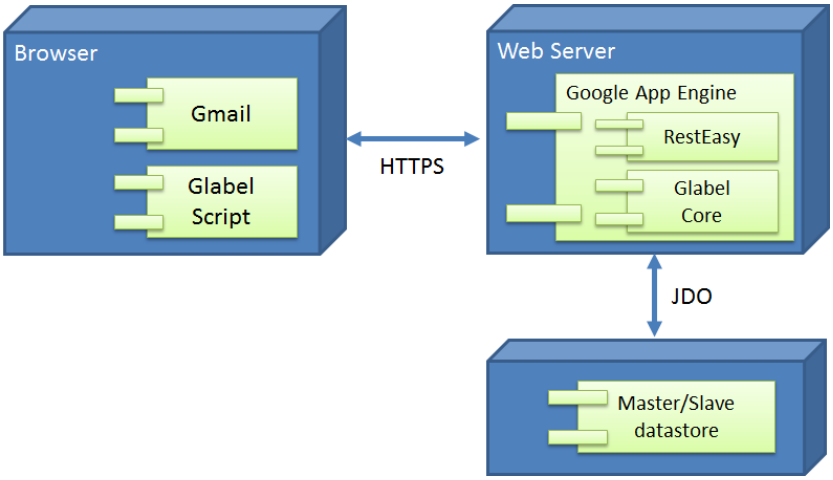


Fig. 2: Glabel Client-Server architecture

The server side was implemented using Google App Engine (GAE), a cloud computing service provided by Google. The App Engine uses the Java Servlet standard for web applications, so any technology designed under this standard, such as Java Server Pages, can be used to implement the web application. The datastore of the App Engine is a non-relational datastore with a search engine supporting atomic transactions. Glabel uses Java Data Objects (JDO) to represent persistent data. JDO is a standard that separates the manipulation of data from the manipulation of the database. This separation of concerns allows a high degree of independence between the view of the data in the model and the view of the data in the database. The communication between the client and the server is implemented using RESTful Web Services. REST (Representational State Transfer) is a set of architectonic principles to design Web Services centered in the resources of the system. These principles include how the different stats of the resources are accessed and transferred using the HTTP protocol.

In the client side, Glabel's Script need to be installed in the Web Browser. This script manipulates Gmail's DOM (Document Object Model) in order to integrate the agent's graphical interface. There is an abstraction layer between Gmail's DOM and Glabel's Script. This layer, named Gmail API, allows Glabel to abstract from the implementation details of the current DOM of Gmail reducing the impact of possible changes in Gmail. It also allow to integrate Glabel to other webmail clients different than Gmail. There are also a set of Event Handler in charge of capturing different user interactions with Gmail Graphical User Interface (GUI). The Event Handlers call different services from the component named Glabel Core API, which is located in a web container within Google Application Engine in the server side. Figure 3 shows the architecture of Glabel in the client side.

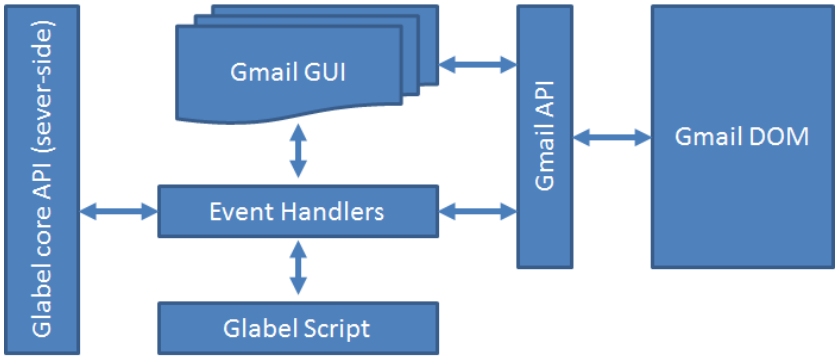
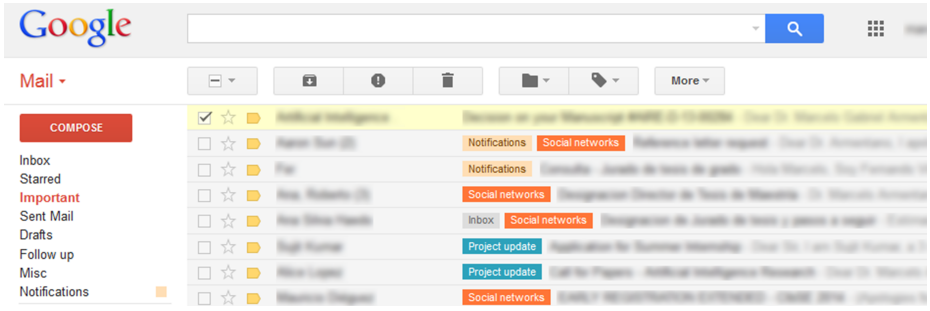


Fig. 3: Glabel Client's architecture

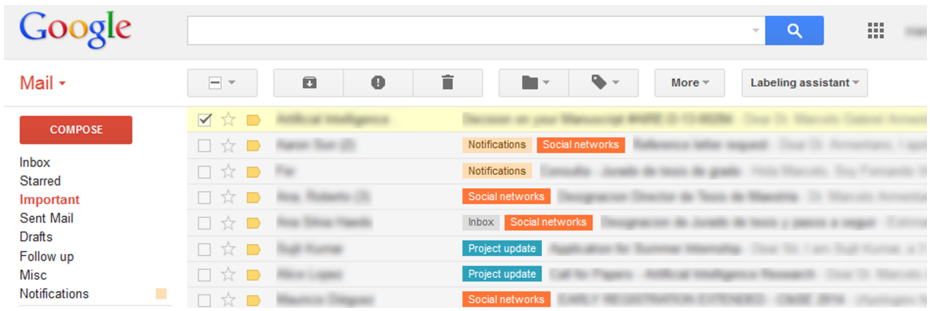
Since the interaction metaphor used to integrate intelligent assistants with conventional applications can affect the user perception of the assistant's capabilities [Armentano et al., 2006], *Glabel* graphical user interface (GUI) was carefully designed in order to result natural to Gmail users. The buttons bar was the place selected to add the labeling assistant. Figure 4 shows Gmail original buttons bar and how the bar is modified to integrate Glabel's GUI using a simple button with the same style than the buttons originally existing in Gmail's interface.

When the user clicks on the "Labeling Assistant" button, the assistant selects the first unread and unlabeled email from the current view, creates a new situation for the decision making algorithm presented in the previous section (Algorithm 5) and waits for the suggested label, type and modality of the interaction. The type of interaction can be (1) a suggestion about the label to apply to the selected email, (2) a warning indicating that the agent has a decision or not, in which case the assistant will ask for explicit feedback about the action to be taken, (3) the automatic labeling of the selected email and (4) do nothing and proceed with the next unread and unlabeled email. Figure 5 shows an example of the result of evaluating the first unread and unlabeled email from the current view of Gmail's interface. Glabel automatically has selected the message and shows a dialog to the user, in this case with a suggestion about the label that could be applied to the selected message.

Figure 6a shows a detail of the options presented to the user along with a assistance suggestion. By clicking on the "OK" button, the user indicates the assistant that the suggestion was useful, but he/she prefers to apply the label himself/herself. This interface also allows the user to accept the suggested label and ask the agent to apply it automatically using the "OK and solve" button. If the user clicks on this button, similar messages might be automatically labeled in the future. The button "NO" indicates the assistant that the suggestion was incorrect. The buttons labeled "<<" and ">>" enable the user to navigate backwards and forwards the messages. Finally, the checkbox in the dialog indicates



(a) Gmail's classical buttons bar



(b) Gmail's buttons bar with Glabel assistant

Fig. 4: Gmail's buttons bar with and without Glabel's user interface

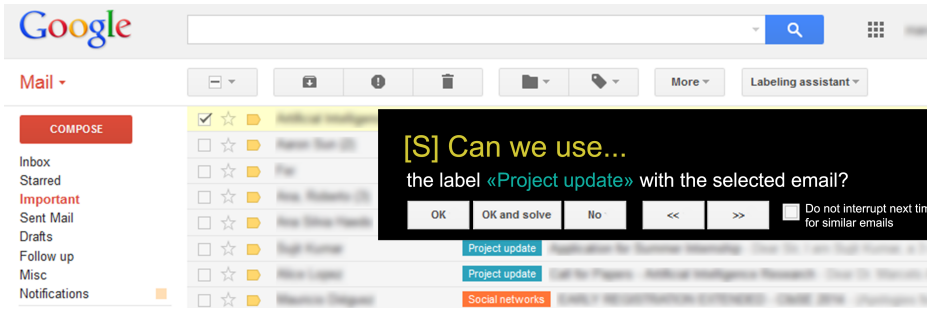


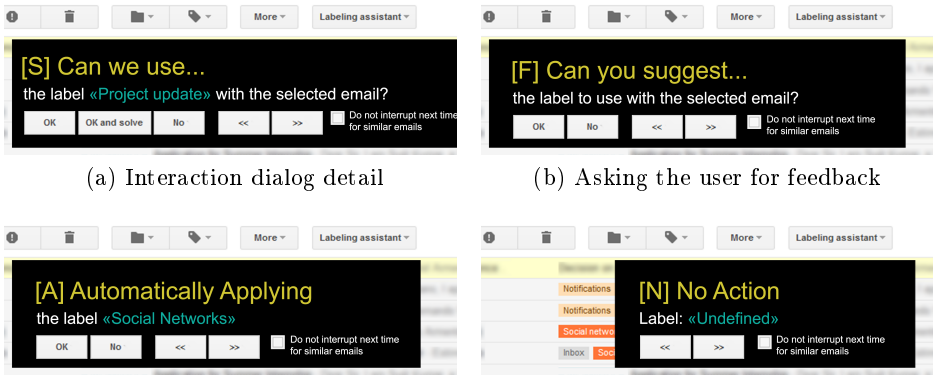
Fig. 5: Glabel interaction with the user

the assistant that the situation is important and that he/she wants to be notified with an interruption the next time a similar situation occurs.

Figure 6b shows a screenshot of Glabel's interface asking the user for feedback when it can not make a decision about which label to apply to a selected message. If the user do provide feedback to the assistant, the given label will be used to tag the selected message automatically and the assistant will learn about this situation. If the user indicates that he/she can not provide a label for the selected email, the assistant will ignore similar situations in the future, unless the observation of the future behavior of the user enables the assistant to infer which label to apply in a similar situation.

Figure 6c shows the dialog that is shown to the user when the assistant has automatically applied a label. The buttons labeled "OK" and "NO" allow the user to provide feedback about the decision taken by the assistant so that it can adapt it future behavior.

Finally, Figure 6d shows the notification dialog that is shown to the user when the assistant decides not to tag an email.



(a) Interaction dialog detail

(b) Asking the user for feedback

(c) Glabel acting automatically on behalf of the user

(d) Ignoring a situation (no action)

Fig. 6: Glabel interaction dialogs

5.2 Participants

The rationale for choosing the participants for the experiment was that they used Gmail daily (so that they were used to its main features), that they receive a large volume of daily emails and that they were used to labeling the incoming emails to organize their inbox. The experiments performed involved thirty eight volunteer users with extensive experience in the use of the Gmail email tool. All participants used Gmail daily as a working tool, being able to use the different

options provided by the email client for sorting email. From the thirty eight users, 14% used less than ten labels to sort their mail, 36% used 10 to 20 labels, 33% used 20 to 30, and the remaining 17% used more than 30 labels. All users receive on average more than one hundred incoming emails per day.

5.3 Experiment iterations

When a user starts using an interface agent, there exist the problem called *cold start*. This means that the agent has no knowledge about the classification habits of the user it will assist. To face this problem *Glabel* starts learning about the user by analyzing the first fifty emails located in his/her inbox having the following two properties: (1) it is a read email and (2) it has an associated label. This way, we make the assumption that labeled read emails are good candidates as training instances since the user has already manually assigned a label to them according to the content of the email.

We first asked each participant to install and configure the assistant, informing them that, when enabled, it was going to build a knowledge base taking into account the emails that he/she receives. We also asked each participant to set the size of each page of the inbox to show fifty emails, allowing the agent to initiate the first analysis (cold start) with the most recent emails.

Then, we started a set of iterations in which we asked each user to:

1. choose ten of the fifty messages and apply the labels previously identified by marking them as read. The other forty messages will remain unlabeled and marked as unread.
2. if the page the user has just analyzed is previous to the fifth page, ask the agent for assistance.
3. confirm whether the assistance provided by the agent fulfilled his/her expectations.
4. for each assistance action, provide the agent with explicit feedback using the agent's graphical interface.
5. go to the next page of the inbox, featuring fifty new unread and unlabeled emails.
6. start again from step 1.

After these iterations, the agent will have gained some knowledge about what type of assistance the user prefers and about the possible labels the user uses for different emails. During the experiment, we automatically collected data in background about the performance of the agent.

5.4 Results

The performance of the agent was evaluated by direct observation: as the user interacted with the agent we registered if the assistance offered was correct or not and computed the accuracy according to the sample size. For each iteration, the set of emails were divided into training instances and testing instances. The

set of training instances is given by the emails on which the agent knows how to act according to it experience in assisting the user. The set of testing instances corresponds to the emails on which the user interacted with the agent to validate it behavior.

We evaluate the accuracy in three dimensions: 1) the content of the assistance (the label to apply to each incoming email), 2) the assistance type selected by the agent (suggestion, warning, action, no-action), and 3) the interruption modality (to interrupt the user or not).

Figure 7, shows the fluctuation of the accuracy of the agent regarding the content of the assistance offered to the user (that is whether the labeled suggested or applied was correct or not) with respect to the number of instances in the training set. Accuracy in this context is computed then as

$$accuracy = \frac{number\,of\,correct\,labeled\,emails}{number\,of\,emails\,labeled}$$

We can see that the accuracy on the label suggested (or applied) to the user ranges between 78.6% and 82.8%, with an average of 79.6% (1.62 standard deviation).

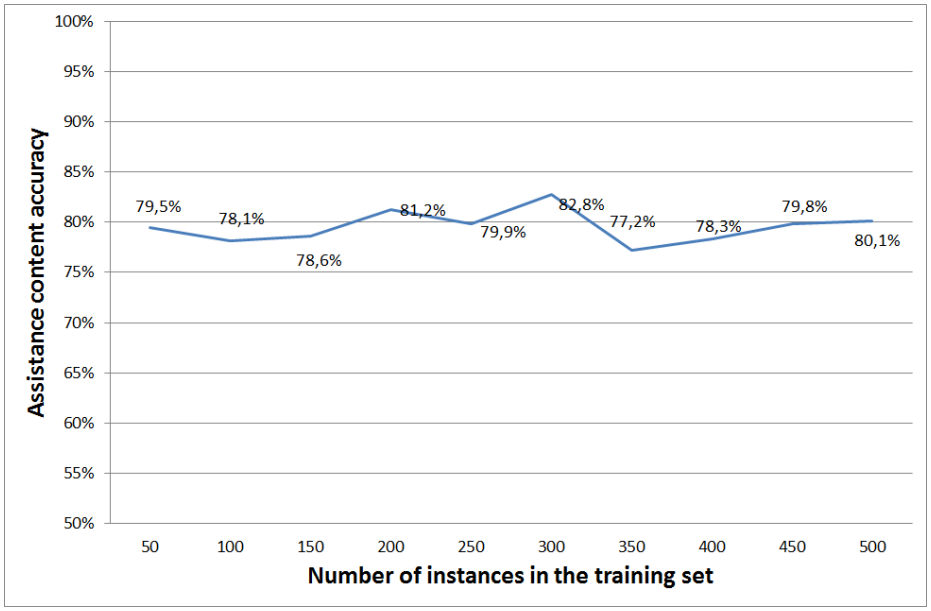


Fig. 7: Accuracy in the assistance content (label)

Figure 7, shows the results of the fluctuation accuracy of the assistance type selected by the agent (that is whether the agent interaction with the user was in the form of a suggestion, a notification, automatically applying a label or doing nothing) with respect to the number of emails in the training set. Accuracy in this context is computed as:

$$accuracy = \frac{number\,of\,correct\,assistance\,type}{number\,of\,interventions}$$

We can observe that the assistant type selected by the agent achieve an excellent performance, ranging between 79.6% and 86.7% with an average of 83.4%. It is worth noticing in this case, that 75.5% of the agent interventions was in the form of a notification, 4.88% in the form of a warning, 8.4% in the form of an automatic labeling action and in 11.1% of the situations the agent decide not to interact with the user.

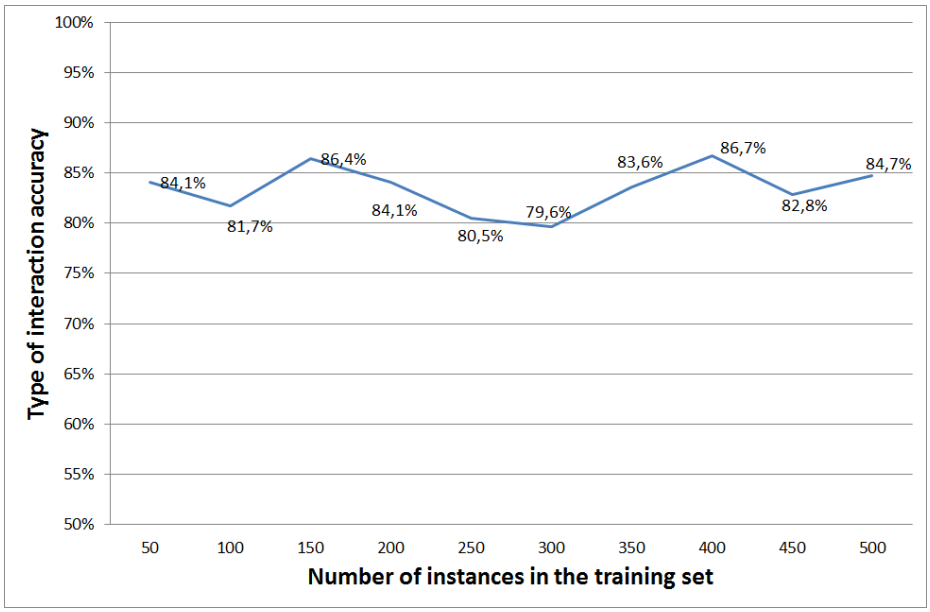


Fig. 8: Accuracy in the assistance type

Finally, we considered the accuracy in the interruption modality, that is if the agent decided to interrupt the user or not. Accuracy in this context is computed then as

$$accuracy = \frac{number\,of\,correct\,interruptions}{number\,of\,interventions}$$

After 500 situations considered by the agent, the agent decided to interrupt 89.4% of the times, with an accuracy of 77.5% and not to interrupt 10.6% of the

times with an accuracy of 86.3%. Figure 9 shows the variation of the accuracy in the interruption modality for each iteration of the experiment.

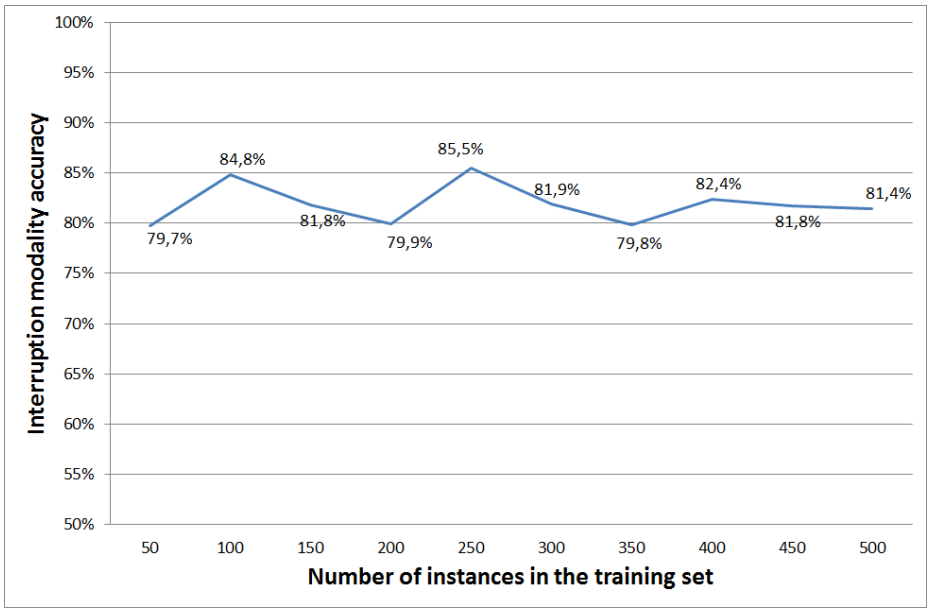


Fig. 9: Accuracy in the interruption modality

Table 1 summarizes the average performance of each classifier.

Classifier	Average Accuracy	Standard Deviation
Content (label)	79.6%	1.62
Type of interaction	83.4%	2.33
Interruption modality	81.9%	1.97

Table 1: Average accuracy of each classifier

Regarding the individual evaluation of each classifier for each iteration of the experiment we can observe that the accuracy is always above 79%.

5.5 Comparison with confidence-based approach

In this section, we compare our approach with the traditional confidence-based approach. In order to make both approaches comparable, we consider the number of interventions of the agent (situations) and the number of correct interventions

of the agent in all aspects (content, interaction type and interruption modality) to compute the accuracy of the agent recommendations:

$$accuracy = \frac{number\,of\,correct\,interventions}{number\,of\,interventions}$$

Figure 10 shows the comparison of the average accuracy of the agent assistance with respect to the user needs, computed at the end of all iterations for the confidence-based approach and the approach followed by *Glabel*. We can see that by using *Glabel,* for 84.8% of the emails the action taken by the agent was the action expected by the user, being this action either a correct suggestion of a label, the request for feedback with an unknown situation, the automatic application of the right label, or simply ignoring an email which the user is not interested in labeling. On the other hand, following the traditional confidence-based approach, the accuracy obtained was 70.1% (14.7% lower). This let us confirm our hypothesis that the simple setting of confidence values is not enough to obtain a satisfying agent-user interaction. The personalization of the type of interaction and the interruption modality in facts enhance the user experience in the labeling task.

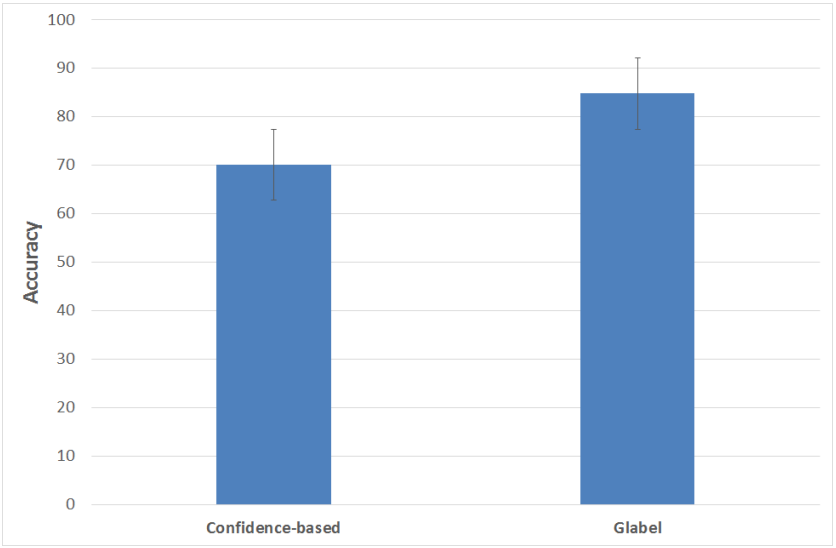


Fig. 10: Comparison of agent accuracy for confidence-based approach and *Glabel's* approach

6 Conclusions

In this work we presented *Glabel*, a labeling assistant that can be integrated with any email client. *Glabel* aims at reducing the tedious task of applying labels to incoming emails by learning about the labels applied by the user in the past. Furthermore, *Glabel* also learns about the type of assistance that the user expects in different situations and also whether the user prefers to be interrupted or not depending on the emails received.

To evaluate our approach, we perform a set of experiments implementing the ideas presented in this article in Google's webmail client, Gmail. These experiments shown an average of 84.8% accuracy of assistant interactions, 14.7% higher than using the traditional confidence-based approach. Although *Glabel* is currently integrated with Gmail, the client-server architecture implemented using RESTful Web Services allows the assistant to be easily integrated with other webmail clients. The cost of this change in the implementation corresponds only to changes in the client script, which allows Glabel to be implemented for other platforms.

The use of *Glabel* enable users to increase their efficiency in the management of emails. Although it is targeted to users who receive a large volume of emails and that are used to the classification mechanisms provided by the email clients, the assistant can be used as an alternative by users who do not have the enough knowledge to build filtering or classification rules. Since *Glabel* accepts different feedback mechanisms, both implicit and explicit, the user can teach the agent about his/her needs and expectations in different situations. The user can then gradually delegate to the assistant the task of sorting his/her emails.

Acknowledgments

We would like to thank Anibal Martín Llano who collaborated with the implementation and evaluation of the agent described in this article.

This research was partially supported by ANPCyT through PICT project No. 2011-0546

Bibliography

Douglas Aberdeen, Ondrej Pacovsky, and Andrew Slater. The learning behind the gmail priority inbox. NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds, December 2010.

Rakesh Agrawal, Roberto J. Bayardo, Jr., and Ramakrishnan Srikant. Athena: Mining-based interactive management of text database. In *Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '00, pages 365–379, London, UK, UK, 2000. Springer-Verlag.

Analia Amandi, Marcelo Campo, Marcelo Armentano, and Luis Berdun. Intelligent agents for distance learning. *Informatics in Education*, 2(2):161–180, 2003.

I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. Spyropoulos. An evaluation of naive Bayesian anti-spam filtering. In *Proc. of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning*, pages 9–17, 2000.

Marcelo Armentano, Daniela Godoy, and Analía Amandi. Personal assistants: Direct manipulation vs. mixed initiative interfaces. *International Journal of Human-Computer Studies*, 64(1):27–35, January 2006.

Marcelo G. Armentano and Analía A. Amandi. The effects of negative interaction feedback in a web navigation assistant. In Masaaki Kurosu, editor, *Human-Computer Interaction. Users and Contexts of Use*, volume 8006 of *Lecture Notes in Computer Science*, pages 107–116. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39264-1.

William W. Cohen. Learning rules that classify e-mail. In *AAAI spring symposium on machine learning in information access*, 1996.

Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '92, pages 318–329, New York, NY, USA, 1992. ACM.

Sarah Jane Delany, Pádraig Cunningham, Alexey Tsymbal, and Lorcan Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4-5):187–195, August 2005. ISSN 0950-7051.

Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, November 1997. ISSN 0885-6125.

Mark Dredze, Bill N. Schilit, and Peter Norvig. Suggesting email view filters for triage and search. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1414–1419, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

H. Drucker, D. Wu, and V. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

Charles Elkan. Boosting and naive bayesian learning. Technical report, Department of Computer Science and Engineering University of California, San Diego, 1997.

Takashi Isozaki, Kazunaga Horiuchi, and Hirotsugu Kashimura. A new e-mail agent architecture based on semi-supervised bayesian networks. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06) - Volume 01*, CIMCA '05, pages 739–744, Washington, DC, USA, 2005. IEEE Computer Society.

Andrew Jennings and Hideyuki Higuchi. A user model neural network for a personal news service. *User Modeling and User-Adapted Interaction*, 3:1–25, 1993.

Che Kao-Li, Tsung-Hsien Yang, and Wei-Po Lee. Personalized multimedia recommendation with social tags and context awareness. In *Proceedings of the World Congress on Engineering 2011 (WCE 2011)*, pages 1046–1051, London, U.K., July 6 - 8 2011.

Svetlana Kiritchenko and Stan Matwin. Email classification with co-training. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '11, pages 301–312, Riverton, NJ, USA, 2011. IBM Corp.

Robyn Kozierok and Pattie Maes. A learning interface agent for scheduling meetings. In *Proceedings of the 1st International Conference on Intelligent User Interfaces*, IUI '93, pages 81–88, New York, NY, USA, 1993. ACM. ISBN 0-89791-556-9.

Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In *IN PROCEEDINGS OF THE TENTH NATIONAL CONFERENCE ON ARTI CIAL INTELLIGENCE*, pages 223–228. MIT Press, 1992.

Scott LeeTiernan, Edward Cutrell, Mary Czerwinski, and Hunter Hoffman. Effective notification systems depends on the user trust. In M Hirose, editor, *Proceedings of Human-Computer Interaction–Interact '01*, pages 684–685, Tokyo, Japan, 2001. IOS Press.

James C. Lester, Sharolyn A. Converse, Susan E. Kahler, S. Todd Barlow, Brian A. Stone, and Ravinder S. Bhogal. The persona effect: affective impact of animated pedagogical agents. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '97, pages 359–366, New York, NY, USA, 1997. ACM.

Henry Lieberman and Earl Wagner. End-user debugging for e-commerce. In *Proceedings of the 8th international conference on Intelligent user interfaces*, IUI '03, pages 257–259, New York, NY, USA, 2003. ACM.

Pattie Maes. Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40, July 1994.

Giuseppe Manco, Elio Masciari, and Andrea Tagarelli. A framework for adaptive mail classification. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '02, pages 387–, Washington, DC, USA, 2002. IEEE Computer Society.

H.M. McBreen and M.A. Jack. Evaluating humanoid synthetic agents in e-retail applications. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 31(5):394 –405, sep 2001.

Sun Park and Dong Un An. Automatic e-mail classification using dynamic category hierarchy and semantic features. 2010;27:478-92. *IETE Technical Review*, 27(6):478–492, 2010.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *AAAI-98 Workshop on Learning for Text Categorization*, pages 55–62, 1998.

Georgios Sakkis, Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. Stacking classifiers for anti-spam filtering of e-mail. In L. Lee and D. Harman, editors, *Proceedings of the 6th Conference on Empirical Methods in Natural Language Processing (EMNLP 2001)*, pages 44–50. Carnegie Mellon University, 2001.

Silvia Schiaffino, Marcelo Armentano, and Analia Amandi. Building respectful interface agents. *International Journal of Human-Computer Studies*, 68(4): 209–222, April 2010.

Richard B. Segal and Jeffrey O. Kephart. Mailcat: an intelligent assistant for organizing e-mail. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 925–926, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

Song Shaoyun and Ma Yu. Intelligent agent-based web learning system. In *IT in Medicine and Education (ITME), 2011 International Symposium on*, volume 2, pages 340–343, dec. 2011.

Juan Francisco Silva Logroño, Luis Berdún, Marcelo G. Armentano, and Analia Amandi. Discrete sequences analysis for detecting software design patterns. In *Advances in New Technologies, Interactive Interfaces and Communicability*, volume 7547 of *Lecture Notes in Computer Science*, pages 197–207. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34009-3.

K. Yelupula and Srini Ramaswamy. Social network analysis for email classification. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ACM-SE 46, pages 469–474, New York, NY, USA, 2008. ACM.

Shinjae Yoo, Yiming Yang, and Jaime Carbonell. Modeling personalized email prioritization: classification-based and regression-based approaches. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 729–738, New York, NY, USA, 2011. ACM.

Seongwook Youn and Dennis McLeod. Spam email classification using an adaptive ontology. *Journal of Software*, 2(3):43–55, September 2007.

Bo Yu and Dong-hua Zhu. Combining neural networks and semantic feature space for email classification. *Knowledge-Based Systems*, 22(5):376–381, July 2009. ISSN 0950-7051.